Best Robotics

Sample Program Quick Start

Overview

The documents describe the program "Best Competition Template.c" which contains the sample source code to control a typical robot for the 2010 BEST competition. A second file -- "Best_Function_Library.h" -- is also needed; it contains library functions for controlling motors and joysticks via the VEX Game Controller.

Getting Started

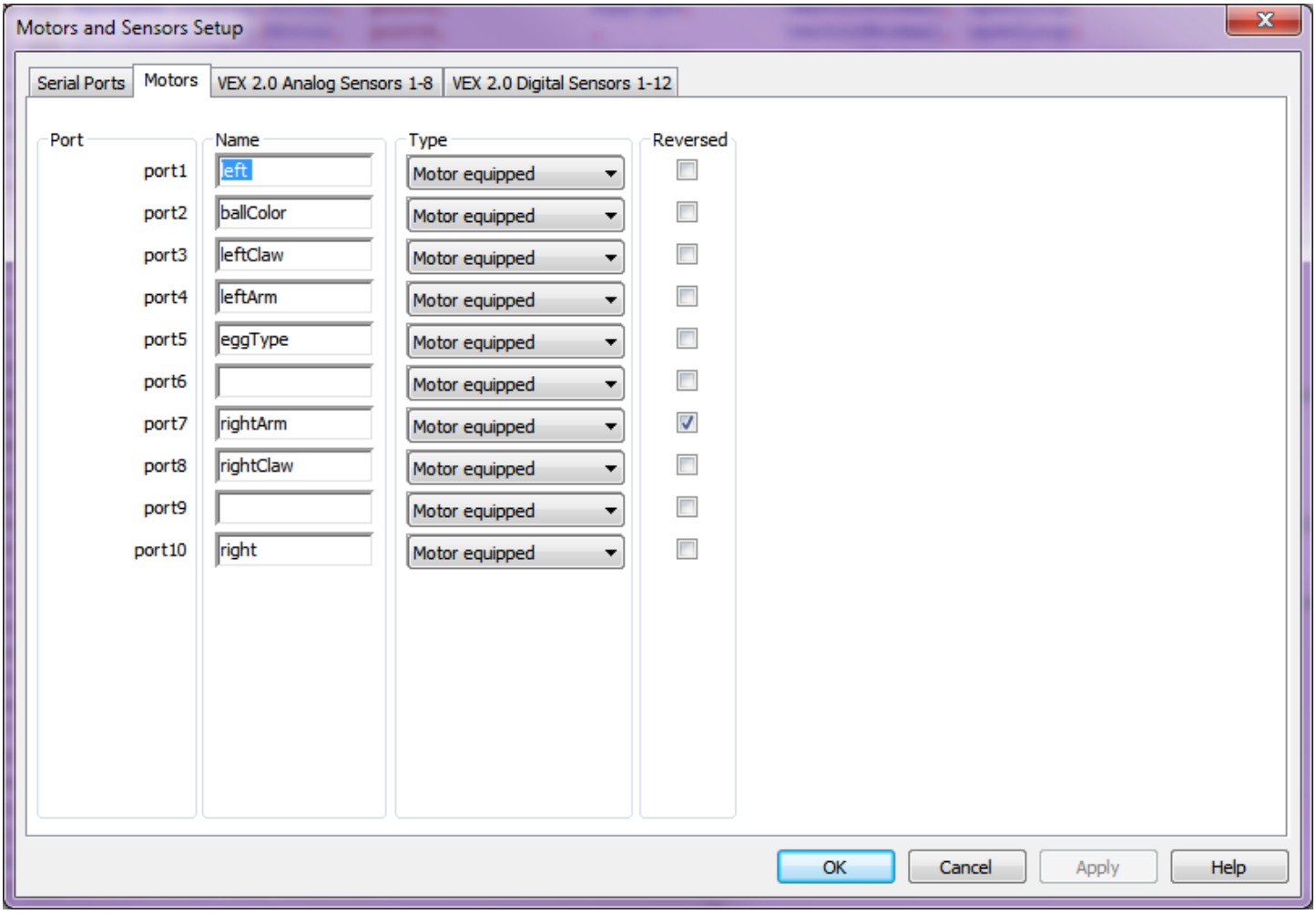
1. COmpletely skim this file. If the program does not work with your robot you may need to make minor changes.
2. Copy the two files "Best Competition Template.c" and "Best_Function_Library.h" to a file directory of your choice.
3. Ensure that you have the latest version of ROBOTC installed. At time of writing this was version 2.30. Version 2.30 has important changes that improve the communications link between the VEX controller and your PC.
4. Start ROBOTC application.
5. If you've just installed version 2.30 you'll have to update the firmware in
    a. The VEX Game Controller
    b. The Cortex controller Master CPU.
    c. The Cortex controller User CPU.

    . ROBOTC has built-in commands for doing this under the "Robot" menu.
6. Ensure that the "Platform type" is set to "VEX 2.0". ROBOTC supports both the original "VEX PIC and the newer "VEX Cortex" in a single version.
7. Use the "Open and Compile" command to open the file "Best Competition Template.c". You may want to save as a different name so that you don't modify the original.
8. Use the "Compile and Download" command to compile your program and store it in the VEX Cortex.
9. You're ready to go and start testing your robot.
10. You may have to make minor edits to the sample program if your robot does not exactly match the configuration of the sample robot.

Setting Up the Motors

Use the following screen to verify and set up the wiring for your various motors. It's found from the ROBOTC menu "Robot -> Motors and Sensors Setup" and then select the "Motor" tab. The symbolic names assigned to a motor -- e.g. "left" -- are used by the main program to determine which motor ports control the different servos and motors. If you change the name here your program will no longer compile.

From this screen you can tell which motor functions are mapped to each of the VEX communications port.



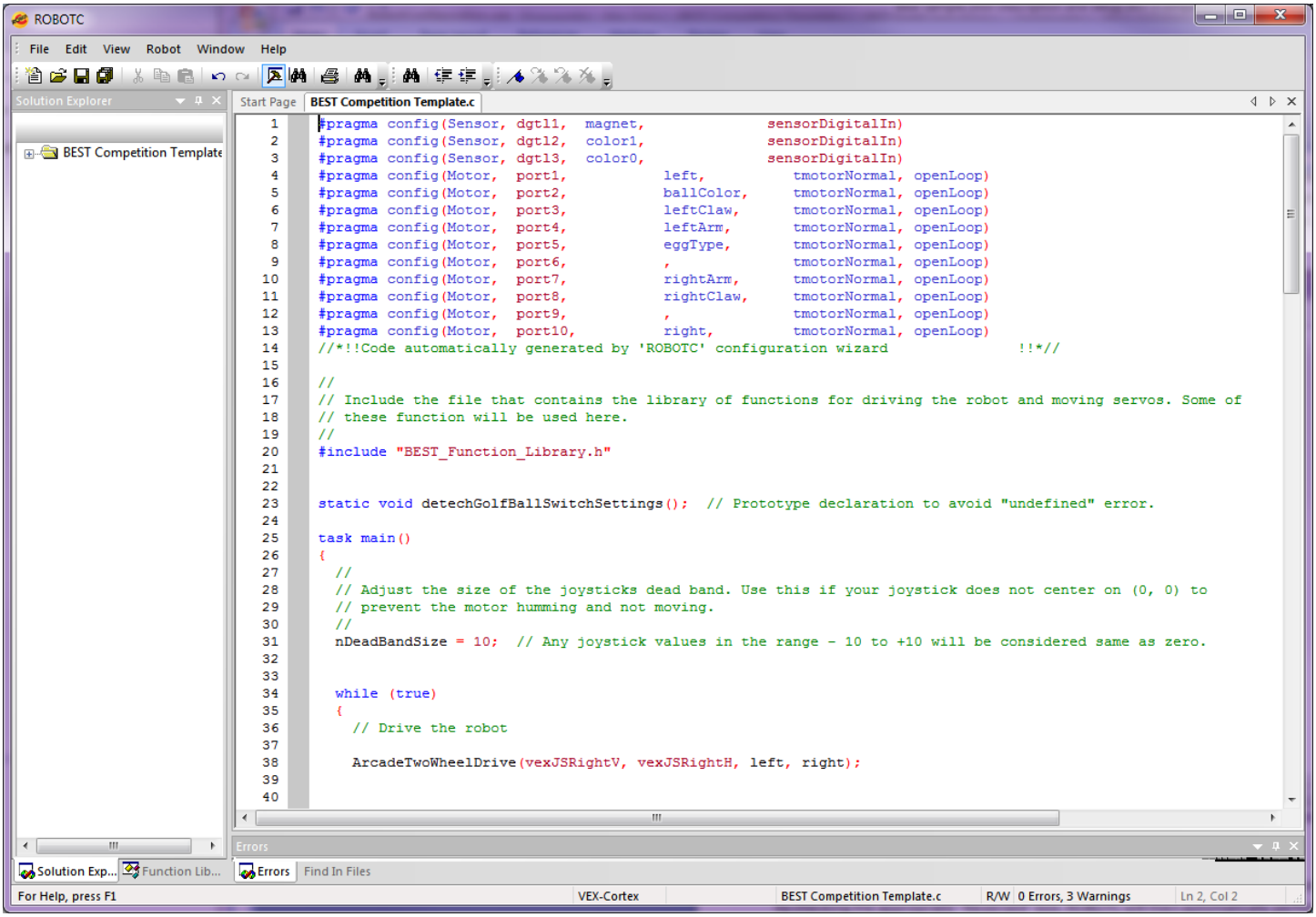
Notes:

1. Suppose during your testing you find that a motor is moving the opposite of what you expect. You can simply fix this by checking the appropriate "Reversed" box. ROBOTC will then automatically perform the correction after you recompile and redownload your program.

2. You might find that the above does not exactly align with your wiring. This is just a illustrative sample. You could rewire your robot to match the above. But its far easier to just change the

Name field. Suppose the servo showing the golf ball color was really wired to port 9; change the name of port2 to blanks and change the name of port 9 to "ballColor". Cut and paste will work.

The motors and sensors setup is just a convenient graphical "editor" for the following statements found at the beginning of your source file.

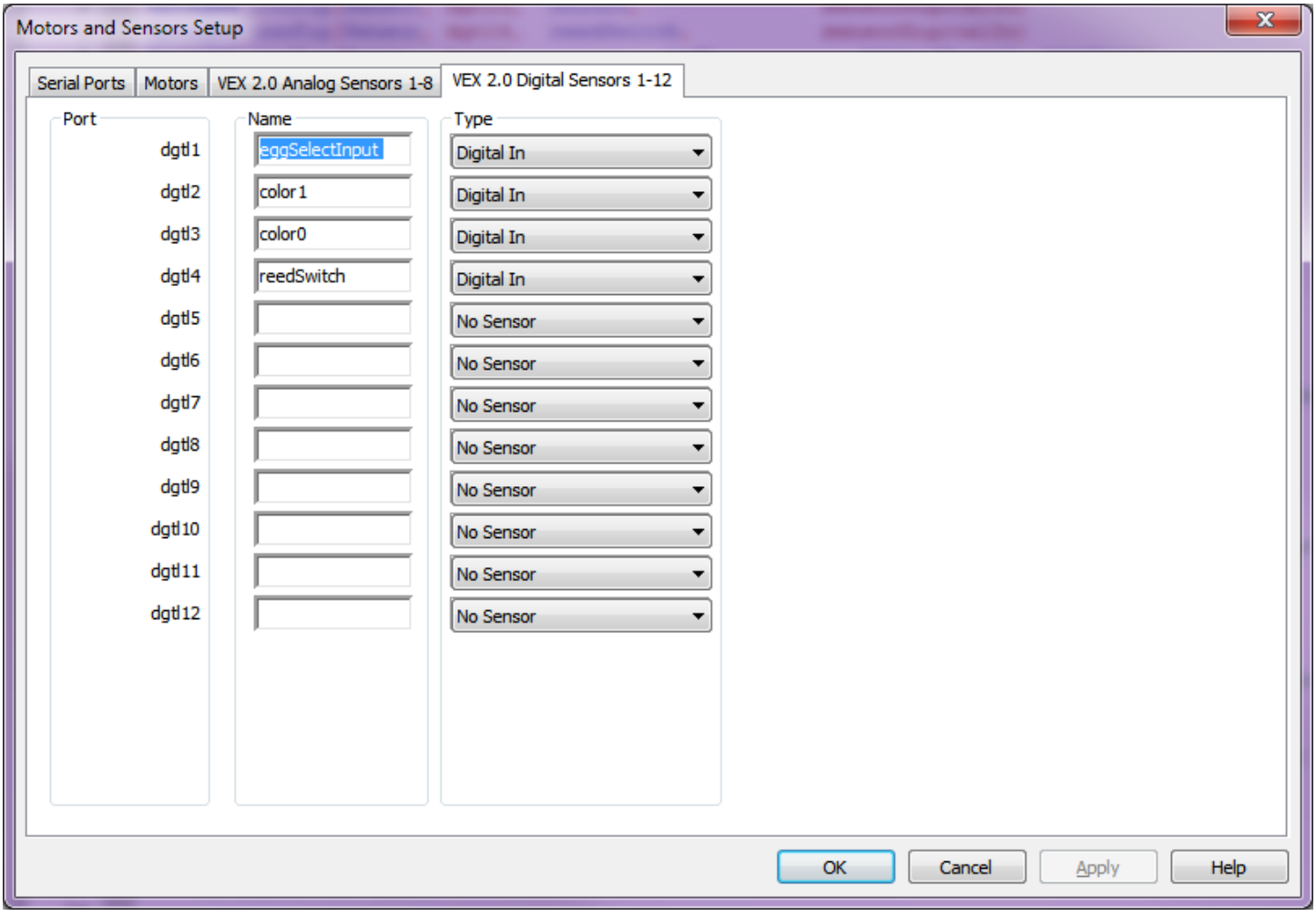


Setting Up the Digital Inputs

The 2010 competition has three "sensors" (digital inputs) for:

- One input is used for determining the egg type (with and without magnet inside) that you need to collect.
- Two inputs are used for determining the color of ball that should be rejected.
- Some teams will also use a fourth input connected to a reed switch to determine whether a egg contains a magnet. There are other techniques than the reed sensor for checking for this.

You need to configure ROBOTC for these inputs. Use the "Robot -> Motors and Sensors Setup" command that you used previously to see how they are configured. Select the "Digital Sensors" tab. You'll get a screen like the following.



In this example, the first three digital inputs correspond to the three switches on the Game Field. The fourth input corresponds to the reed switch if used.

If you've wired your sensors differently then you'll have to edit this screen. Make sure to not misspell the names or your program will no longer compile -- the names are used in the source code for your program as well.

If you don't feel comfortable editing this screen, then wire your inputs to correspond to the inputs on the screen!

Setting Up to Drive the Robot

The sample program is set up to drive the robot using "Arcade style" driving with 2-wheel drive. A single two axis joystick is used. The vertical direction controls the speed and the horizontal direction controls the turning. The code for this is

```
// Drive the robot

ArcadeTwoWheelDrive(vexJSLeftV, vexJSLeftH, left, right);
//TankTwoWheelDrive(vexJSRightV, vexJSLeftV,  left, right);
```

vexJSLeftV and **vexJSLeftH** are symbolic names indicating the two joystick axes used to controlling speed and direction. In this case they indicate that the VEX Game Controller left joystick is used for driving. Suppose you wanted to use the right joystick. You'd just replace them with **vexJSRightV** and **vexJSRightH**.

This example uses "Arcade style" driving. Some teams may prefer tank style. This is easy to configure. Simply comment out the code for the "arcade" and uncomment the code for "tank". It would now become.

```
// Drive the robot

//ArcadeTwoWheelDrive(vexJSLeftV, vexJSLeftH, left, right);
TankTwoWheelDrive(vexJSLeftV, vexJSRightV,  left, right);
```

For tank style robots the two vertical axes -- one for the left (**vexJSLeftV**) and one for right (**vexJSRightV**) joystick are used.

There are also similar functions for four wheel driving. But most BEST teams will only use one motor on each side.

If you find that one of the motors is turning the opposite of what you expect, then this is easy to change. You could reverse the leads to the robot; but this is the most complicated way. It's easier to have ROBOTC look after this in its internal software. Go back to the section on "Setting up the Motors" and read how to do this.

Motor Dead Band Control

Even after your joysticks have been calibrated you may find that they don't always return to the centered (0, 0) position when they are released. They may be off by a few counts. You'll notice this by a humming noise coming from your motors.because they are being provided by a little bit of power which isn't sufficient for them to move. The humming noise is internal vibration in the motor as they try to move. This is easy to correct with a small dead zone; any joystick value in the range -10 to +10 is treated as a zero value for powering the motors.

You may want to adjust the size of the dead band. The sample program uses a dead band of 10.

```
//
// Adjust the size of the joysticks dead band. Use this if your joystick does not
// center on (0, 0) to prevent the motor humming and not moving.
//
nDeadBandSize = 10;  // Joystick values from - 10 to +10 will be treated as zero.
```

If you change the "10" to "15" you'll increase the dead band size.

```
//
// Adjust the size of the joysticks dead band. Use this if your joystick does not
// center on (0, 0) to prevent the motor humming and not moving.
//
nDeadBandSize = 15;  // Joystick values from - 15 to +15 will be treated as zero.
```

Controlling A Servo via Game Controller Buttons

You'll want to be able to control a servo's position using the VEX Game Controller. This is typically done with two buttons. One moves the servo down and the second moves the servo up. There are two popular control algorithms for this.

1. Tap one of the up or down buttons and have the servo immediately move to one of its two extreme positions.
2. When a button is held down, the servo moves in the direction for that button. When you release the button the servo holds the current position.

A single ROBOTC library function is used for both algorithms. Here's some of the code from the sample program.

```
// Move the Left arm motor
MotorControlViaDigitalButtons(Btn7D,      // The joystick button that controls speed decrease
                              Btn7U,      // The joystick button that controls speed increase
                              leftArm,    // The motor that is being controlled
                              -127,       // The lowest  value for the motor speed
                              +127,       // The highest value for the motor speed
                              0);         // How fast to adjust the speed. 0 -- fastest.
                                          // 50 -- 1.25 seconds to cover the compete range.

// Open and close the Left claw servo
ServoControlViaDigitalButtons(Btn7L,      // The joystick button that controls downward movement
                              Btn7R,      // The joystick button that controls upward movement
                              leftClaw,   // The servo that is being controlled
                              -127,       // The lowest DOWN value for the servo position
                              +127,       // The highest UP value for the servo position
                              0);
```

These two function calls are used to control and lifting arm and associated grabber / "claw".

- Function **`MotorControlViaDigitalButtons`** controls the motor on a lifting arm. The lifting arm is controlled by motor "**`leftArm`** " which is the same as motor port 4 -- go back to the section on "Setting Up the Motors" to see how this association is set up. If your lifting arm uses a servo then use function **`ServoControlViaDigitalButtons`** instead.

- Function **`ServoControlViaDigitalButtons`** controls a servo to open and close a claw. The claw is controlled by servo **`leftClaw`** which is the same as motor port 3.

Let's look at how the claw is controlled first (i.e. the second function call) since it uses a servo.

The LEFT and RIGHT buttons for group 7 control the claw; that's the constants **`Btn7L`** and **`Btn7R`** in the source code. Change these if you want to use a different pair of buttons. For example, if you wanted to use the "8" group of buttons then change the four variables for buttons from "**`Btn7D, Btn7U, Btn7L, Btn7R"`** to **`"Btn8D, Btn8U, Btn8L, Btn8R"`** in the above function calls.

The two extreme positions of the servo are value "-127" and "+127' which are the maximum values for a servo. If you find that this is too much you can reduce them to smaller values. Suppose you want to change them to "-100" and "+90". The function call then becomes.

```
ServoControlViaDigitalButtons(Btn7L,      // The joystick button that controls downward movement
                              Btn7R,      // The joystick button that controls upward movement
                              leftClaw,   // The servo that is being controlled
                              -100,       // The lowest value for the servo position
                              +90,        // The highest value for the servo position
                              0);         // How fast to adjust the speed. 0 -- fastest. 50 -- 1.25
                                          // seconds to cover the compete range.
```

Now you find that the buttons are moving the servo in the opposite direction to what you desire. Swap the order of the two parameters. So that instead of "-100, +90" they now become "+90, -100". Now the movement will be what's desired.

```
ServoControlViaDigitalButtons(Btn7L,      // The joystick button that controls downward movement
                              Btn7R,      // The joystick button that controls upward movement
                              leftClaw,   // The servo that is being controlled
                              +90,        // The lowest value for the servo position
                              -100,       // The highest value for the servo position
                              0);         // How fast to adjust the speed. 0 -- fastest. 50 -- 1.25
                                          // seconds to cover the compete range.
```

The last parameter in the example has value zero. This indicates that the servo should immediately move to one of the extreme positions as soon as one of the buttons is pushed. If its non-zero, then it tells how fast the servo position should be adjusted. A value of 10 will cause the servo to take 0.25 seconds to move over its full range; a value of 50 will take five times as long -- 1.25 seconds.

The sample program illustrates control over two separate lifting arms. Your robot may only have one. You can delete the source code for the second arm.

Controlling A Motor via Game Controller Buttons

You'll also want to be able to control a motor's position using the VEX Game Controller. buttons instead of a joystick. Sometimes you run out of joystick axes!

Control of a motor via buttons is similar to controlling a servo. Except that when you release both buttons you'll want the motor to stop; i.e. it's speed value is set to zero. With a servo you want it to hold its current position.

Just like servos, there's a need for two algorithms for controlling a motor.

1. As soon as a button is pushed, immediately run the motor at the maximum speed for the button.

2. Gradually ramp the motor to its maximum speed. This enables a more precise positioning with the motor.

Let's go back to the sample program control of the lifting arm. It's the following function call.

```
// Move the Left arm motor
MotorControlViaDigitalButtons(Btn7D,      // The joystick button that controls speed decrease
                              Btn7U,      // The joystick button that controls speed increase
                              leftArm,    // The motor that is being controlled
                              -127,       // The lowest  value for the motor speed
                              +127,       // The highest value for the motor speed
                              0);         // How fast to adjust the speed. 0 -- fastest.
                                          // 50 -- 1.25 seconds to cover the compete range.
```

The UP and DOWN buttons for group 7 control the arm motor; that's the constants **`Btn7D`** and **`Btn7U`** in the source code. Change these if you want to use a different pair of buttons. For example, if you wanted to use the "8" group of buttons then change the four variables for buttons from "**`Btn7D, Btn7U"`** to **`"Btn8D, Btn8U"`** in the above function calls. Make a similar change for the claw servo.

The two extreme positions of the motor speed are value "-127" and "+127' which are the maximum possiblevalues. If you find that these are too fast you can reduce them to smaller values.

If you find the motor is running opposite to what you expect then go back to the "Setting Up The Motors" section to find out how to reverse the motor movement.

Sampling the Egg Type and Reject Color

The sample program contains a possible implementation of the code to sample the egg type and reject color inputs for the 2010 BEST competition. And then display the results via servo outputs. I'll leave it as an exercise to the reader to understand the logic.